SciSpot: Scientific Computing On Temporally Constrained Cloud Preemptible VMs

JCS

Abstract—Scientific computing applications are being increasingly deployed on cloud computing platforms. Transient servers can be used to lower the costs of running applications on the cloud. However, the frequent preemptions and resource heterogeneity of these transient servers introduces many challenges in their effective and efficient use. In this paper, we develop techniques for modeling and mitigating preemptions of transient servers, and present SciSpot, a software framework that enables low-cost scientific computing on the cloud. SciSpot deploys applications on Google Cloud Preemptible Virtual Machines that exhibit temporally constrained preemptions: VMs are always preempted in a 24 hour interval. Empirical analysis shows that the preemption rate is bath-tub shaped, which raises multiple challenges in modeling and policy design. We develop a new reliability model for temporally constrained preemptions, and use statistical mechanics to show why the bath-tub shape is a fundamental characteristic.

SciSpot's design is guided by our observation that many scientific computing applications (such as simulations) are deployed as "bag" of jobs, which represent multiple instantiations of the same computation with different physical and computational parameters. For a bag of jobs, SciSpot finds the optimal transient server on-the-fly, by taking into account the price, performance, and preemption rates of different servers. SciSpot reduces costs by $5\times$ compared to conventional cloud deployments, and reduces makespans by up to $10\times$ compared to conventional high performance computing clusters.

1 Introduction

Transient cloud computing is an emerging and popular resource allocation model used by all major cloud providers, and allows unused capacity to be offered at low costs as preemptible virtual machines. Transient VMs can be unilaterally revoked and preempted by the cloud provider, and applications running inside them face fail-stop failures. To expand the usability and appeal of transient VMs, many systems and techniques have been proposed that ameliorate the effects of preemptions and reduce the computing costs of applications. Fault-tolerance mechanisms [49], [41], resource management policies [48], [62], and cost optimization techniques [24], [50] have been developed for a wide range of applications—such as interactive web services, distributed data processing, parallel computing, etc.

Transiency mitigation techniques all depend on probabilistic estimates of when and how frequently preemptions occur. For instance, many fault-tolerance and resource optimization policies are parametrized by the mean time to failure (MTTF) of the transient VMs. The preemption characteristics are governed by the transient availability model chosen by the cloud provider.

Spot markets (used by Amazon EC2's spot instances and others) are a popular model, where preemptions are governed by dynamic prices (which are in turn set using a continuous second-price auction [16]). In this paper, we

checkpointing [46], [41], diversification [48], *all* use price-signals to model the availability and preemption rates of spot instances. With flat pricing, these approaches are not applicable. Furthermore, no other information about preemption characteristics is publicly available, not even coarse-grained metrics such as MTTFs. To address this, we develop an *empirical* approach for understanding and modeling preemptions. We conduct a large empirical study of over 800 preemptions of Google Preemptible VMs, and develop an analytical probability model for temporally constrained preemptions.

Due to the temporal constraint on preemptions, classical models that form the basis of preemption modeling and policies, such as memoryless exponential failure rates, are not applicable. We find that preemption rates are *not* uniform, but bathtub shaped with multiple distinct temporal phases, and are incapable of being modeled by existing bathtub distributions such as Weibull. We capture these characteristics by *developing a new probability model*. Our model uses reliability theory principles to capture the 24-hour lifetime of VMs, and generalizes to VMs of different resource capacities, geographical regions, and across different temporal domains. Using our probability model, we find that bathtub failures can reduce the recomputation overhead of preeemptions by more than $10\times$ compared to uniform failures—which has important implications for cloud users and providers.

We show the applicability and effectiveness of our model by developing optimized policies for job scheduling and checkpointing. These policies are fundamentally dependent on empirical and analytical insights from our model. Our job-scheduling policy uses the bathtub behavior to decide whether to run a new job on a running VM or to request a new VM, and reduces job-failure probability by $2\times$ compared to conventional memoryless policies. The bathtub distribution also requires a new approach to periodic checkpointing—since existing Young-Daly [22] checkpointing is restricted to memoryless preemptions. Our checkpointing policy combines our preemption model and dynamic programming to reduce the checkpointing overhead by more than $5\times$. These optimized policies are a building block for transient computing systems and reducing the performance degradation and costs of preemptible VMs. We implement and evaluate these policies as part of a batch computing service, which we also use for empirically evaluating the effectiveness of our model and policies under real-world conditions.

Towards our goal of developing a better understanding of constrained preemptions, we make the following contributions:

1) We conduct a large-scale, first of its kind empirical study of preemptions of Google's Preemptible VMs [1]. We then

3) Based on our preemption model, we develop optimized policies for job scheduling and checkpointing that minimize the total time and cost of running applications. These policies reduce job running times by up to $2\times$ compared to existing preemption models used for transient VMs.
4) We implement and evaluate our policies as part of a batch computing service for Google Preemptible VMs. Our service is especially suitable for scientific simulation applications, and can reduce computing costs by $5\times$ compared to conventional cloud deployments, and reduce the performance overhead of preemptible VMs to less than 3%.

2 Background

We now give an overview of transient cloud computing, and the use of preemption models in transient computing systems.

2.1 Transient Cloud Computing

Infrastructure as a service (IaaS) clouds such as Amazon EC2, Google Public Cloud, Microsoft Azure, etc., typically provide computational resources in the form of virtual machines (VMs), on which users can deploy their applications. Conventionally, these VMs are leased on an "on-demand" basis: cloud customers can start up a VM when needed, and the cloud platform provisions and runs these VMs until they are shut-down by the customer. Cloud workloads, and hence the utilization of cloud platforms, shows large temporal variation. To satisfy user demand, cloud capacity is typically provisioned for the *peak* load, and thus the average utilization tends to be low, of the order of 25% [60], [20].

To increase their overall utilization, large cloud operators have begun to offer their surplus resources as low-cost servers [2]with *transient* availability, which can be preempted by the cloud operator at any time (after a small advance warning). These preemptible servers, such as Amazon Spot instances [2], Google Preemptible VMs [5], and Azure batch VMs [13], have become popular in recent years due to their discounted prices, which can be 7-10$\times$ lower than conventional non-preemptible servers. Due to their popularity among users, smaller cloud providers such as Packet [6] and Alibaba [1] have also started offering transient cloud servers.

However, effective use of transient servers is challenging for applications because of their uncertain availability [53]. Preemptions are akin to fail-stop failures, and result in loss of the application memory and disk state, leading to downtimes for interactive applications such as web services, and poor throughput for batch-computing applications. Consequently, researchers have explored fault-tolerance techniques such as checkpointing [46], [41], [55] and resource management techniques [48] to ameliorate the effects of preemptions. The effect of preemptions depends on the application's delay insensitivity and fault model, and mitigating preemptions for different applications remains an active research area [35].

1. Preemption dataset available at https://github.com/kadupitiya/goog-preemption-data/
2. We use servers and VMs interchangeably throughout the paper.

2.2 Modeling Preemptions of Transient VMs

Underlying *all* techniques and systems in transient computing is the notion of using some probabilistic or even a deterministic model of preemptions. Such a preemption model is then used to quantify and analyze the impact of preemptions on application performance and availability; and to design model-informed policies to minimizing the effect of preemptions. For example, the preemption rate or MTTF (Mean Time To Failure) of transient servers has found extensive use in selecting the appropriate type transient server for applications [48], [55], determining the optimal checkpointing frequency [46], [41], [29], [25], etc.

Preemptions of spot market based VMs (such as EC2 spot instances) are based on their *price*, which is dynamically adjusted based on the supply and demand of cloud resources. Spot prices are based on a continuous second-price auction, and if the spot price increases above a pre-specified maximum-price, then the server may be preempted [16]. Thus, the time-series of these spot prices can be used for understanding preemption characteristics such as the frequency of preemptions and the "Mean Time To Failure" (MTTF) of the spot instances. Publicly available [33] historical spot prices have been used to characterize and model spot instance preemptions [49], [71], [51], [63]. For example, past work has analyzed spot prices and shown that the MTTFs of spot instances of different hardware configurations and geographical zones range from a few hours to a few days [64], [44], [63], [15], [65]. Spot instance preemptions can be modeled using *memoryless* exponential distributions [71], [47], [46], [25], [69], which permits optimized periodic checkpointing policies such as Young-Daly [22].

However, using pricing information for preemption modeling is *not* a generalizable approach, and is not applicable to other models of transient availability used by other transient VMs like Google Preemptible VMs and Azure Low-priority batch VMs. These VMs have *flat* pricing, and thus pricing cannot be used to infer preemptions. Moreover, these cloud providers (Google and Azure) do not expose *any* public information about their preemption characteristics, even coarse grained metrics like MTTF that can be useful in mitigating preemptions [69]. In this paper, we propose an empirical approach for modeling preemptions of temporally constrained VMs such as Google Preemptible VMs. Our empirical data and the resulting preemption model allows the development of preemption mitigation policies. Google Preemptible VMs have a maximum lifetime of 24 hours, and this *constrained* preemption is not memoryless, and requires new fundamental modeling approaches.

3 Understanding temporally constrained VM preemptions

In our quest to understand temporally constrained preemptions, we conduct an empirical study of preemptions of Google Preemptible VMs. Based on our observations and insights from the study, we then develop a probability model for temporally constrained preemptions, which we later use to develop preemption-mitigating resource management application policies.

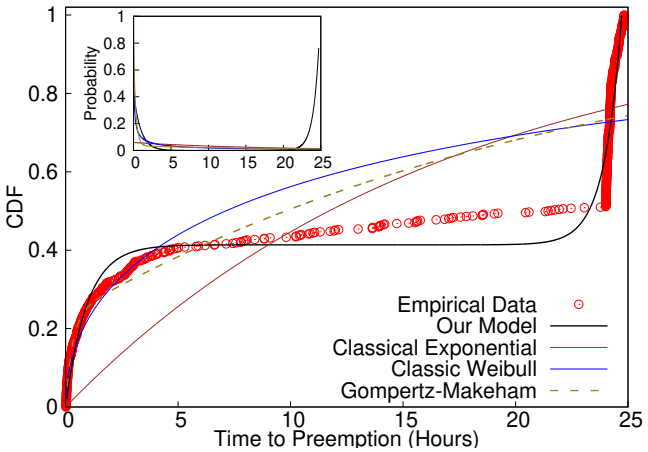

Fig. 1: CDF of lifetimes of Google Preemptible VMs. Our proposed distribution for modeling the constrained preemption dynamics provides a better fit to the empirical data compared to other failure distributions. Inset shows the probability density functions.

3.1 Empirical Study Of Preemptions

Methodology. We launched 870 Google Preemptible VMs of different types over a two month period (Feb–April 2019), and measured their time to preemption (i.e., their useful lifetime). VMs of different resource capacities were launched in a four geographical regions; during days and nights and all days of the week; and running different workloads[3]. We launched VMs in their default resource configurations (CPU and memory), and do not use custom VM sizes. To ensure the generality of our empirical observations, VMs were not launched during well-known peak utilization days (such as Black Friday). The preemption data collection was bootstrapped: a small amount of data points were used to estimate and model the preemption CDF, which we then used to run our batch computing service (described and evaluated in Sections 5 and 6), which generated the rest of the preemption data. Due to the relatively high preemption rates compared to EC2 spot instances, we were able to collect these data points for less than $5,000.

A sample of over 100 such preemption events are shown in Figure 1, which shows cumulative distribution function (CDF) of the lifetime of the `n1-highcpu-16` VM in the `us-east1-b` zone. Our empirical approach allows us to make the following observations.

Significance of bathtub preemptions. The above empirical observations indicate that temporally constrained preemptions are *not* uniformly distributed. The bathtub shaped preemption distribution is not a coincidence. It is a result of fundamental characteristics of constrained preemptions that benefit applications. For applications that do not incorporate explicit fault-tolerance (such as checkpointing), early preemptions result in less wasted work than if the preemptions were uniformly distributed over the 24 hour interval. Furthermore, the low rate of preemptions in the middle periods allows jobs that are smaller than 24 hours to finish execution with only a low probability of failure, once they survive the initial preemption phase. We compare application performance with bathtub preemptions and uniformly distributed preemptions later in Section 6, and find that bathtub preemptions can reduce the performance overheads of preemptions by up $10\times$. However, effective policies for constrained preemptions requires a probability model of preemptions, which is challenging due to the temporal constraint and the steep bathtub behavior. Existing preemption models are not applicable, and we present our new model next.

3. Preemption rates can also be affected by number of VMs launched simultaneously, which we limited to between 1 and 10.

3.2 Failure Probability Model

We now develop an analytical probability model for finding a preemption at time t (preemption dynamics) that is faithful to the empirically observed data and provides a basis for developing running-time and cost-minimizing optimizations. Modeling preemptions constrained by a finite deadline raises many challenges for existing preemption models that have been used for other transient servers such as EC2 spot instances. We first discuss why existing approaches to preemption modeling are not adequate, and then present our closed-form probability model and associated reliability theory connections.

3.2.1 Our model

Our failure probability model seeks to address the drawbacks of existing reliability theory models for modeling constrained preemptions. The presence of three distinct phases exhibiting non-differentiable transition points (sudden changes in CDF near the deadline, for example) suggests that for accurate results, models that treat the probability as a step function (CDF as a piecewise-continuous function) could be employed. However, this limits the range of model applicability and general interpretability of the underlying preemption behavior. Our goal is to provide a broadly applicable, continuously differentiable, and informative model built on reasonable assumptions.

We begin by making a key assumption: the preemption behavior arises from the presence of *two* distinct failure processes. The first process dominates over the initial temporal phase and yields the classic exponential distribution that captures the high rate of early preemptions. The second process dominates over the final phase near the 24 hour maximum VM lifetime and is assumed to be characterized by an exponential term that captures the sharp rise in preemptions that results from this constrained lifetime.

Based on these observations, we propose the following general form for the CDF:

$$\boxed{\mathscr{F}(t) = A\left(1 - e^{-\frac{t}{\tau_1}} + e^{\frac{t-b}{\tau_2}}\right)} \tag{1}$$

where t is the time to preemption, $1/\tau_1$ is the rate of preemptions in the initial phase, $1/\tau_2$ is the rate of preemptions in the final phase, b denotes the time that characterizes "activation" of the final phase where preemptions occur at a very high rate, and A is a scaling constant. The model is fit to data for $0 < t < L$, where $L \approx 24$ hours represents the temporal interval (deadline). Combination of the 4 fit parameters (τ_1, τ_2, b, and A) are chosen to ensure that boundary condition $\mathscr{F}(0) \approx 0$ is satisfied. In practice, typical fit values yield $b \approx 24$ hours, $\tau_1 \in [0.5, 1, 5]$, $\tau_2 \approx 0.8$, and $A \in [0.4, 0.5]$.

For most of its life, a VM sees failures according to the classic exponential distribution with failure-rate equal to $1/\tau_1$ – this behavior is captured by the $1-e^{-t/\tau_1}$ term in Equation 1. As VMs get closer to their maximum lifetime imposed by the cloud operator, they are reclaimed (i.e., preempted) at a high rate $1/\tau_2$, which is captured by the second exponential term, $e^{(t-b)/\tau_2}$ of Equation 1. Shifting the argument (t) of this term by b ensures that the exponential reclamation is only applicable near the end of the VM's maximum lifetime and does not dominate over the entire temporal range.

The analytical model and the associated distribution function $\mathscr{F}$ introduced above provides a much better fit to the empirical data (Figure 1) and captures the different phases of the preemption dynamics through parameters τ_1, τ_2, b, and A. These parameters can be obtained for a given empirical CDF using least squares function fitting methods (we use scipy's `optimize.curve_fit` with the dogbox technique [7]). The failure or preemption rate can be derived from this CDF as:

$$f(t) = \frac{d\mathscr{F}(t)}{dt} = A\left(\frac{1}{\tau_1}e^{-t/\tau_1} + \frac{1}{\tau_2}e^{\frac{t-b}{\tau_2}}\right). \tag{2}$$

$f(t)$ vs. t yields a bathtub type failure rate function for the associated fit parameters (inset of Figure 1).

In the absence of any prior work on constrained preemption dynamics, our aim is to provide an interpretable model with a minimal number of parameters, that provides a sufficiently accurate characterization of observed preemptions data. Further generalization of this model to include more failure processes would introduce more parameters and reduce the generalization power.

Expected Lifetime: Our analytical model also helps crystallize the differences in VM preemption dynamics, by allowing us to easily calculate their expected lifetime. More formally, we define the expected lifetime of a VM ($\mathscr{L}$) as:

$$E[\mathscr{L}] = \int_0^L tf(t)\,dt = -A(t+\tau_1)e^{-t/\tau_1} + A(t-\tau_2)e^{\frac{t-b}{\tau_2}}\Big|_0^L \tag{3}$$

where $f(t)$ is the rate of preemptions of the VM (Equation 2).

This expected lifetime can be used in lieu of MTTF, for policies and applications that require a "coarse-grained" comparison of the preemption rates of servers of different types, which has been used for cost-minimizing server selection [46].

3.2.2 Reliability Analysis

We now analyze and place our model in a reliability theory framework.

Expected Lifetime: Our analytical model also helps crystallize the differences in VM preemption dynamics, by allowing us to easily calculate their expected lifetime. More formally, we define the expected lifetime of a VM ($\mathscr{L}$) as:

$$E[\mathscr{L}] = \int_0^L tf(t)\,dt = -A(t+\tau_1)e^{-t/\tau_1} + A(t-\tau_2)e^{\frac{t-b}{\tau_2}}\Big|_0^L \tag{4}$$

where $f(t)$ is the rate of preemptions of the VM (Equation 2).

This expected lifetime can be used in lieu of MTTF, for policies and applications that require a "coarse-grained" comparison of the preemption rates of servers of different types, which has been used for cost-minimizing server selection [46].

Hazard Rate: The hazard rate $\lambda(t)$ governs the dynamics of the failure (or survival) processes. It is generally defined as $\lambda(t) = \frac{f(t)}{S(t)}$, often expressed via the following differential equation (rate law):

$$\frac{dS(t)}{dt} = -\lambda(t)S(t) \tag{5}$$

where $S(t) = 1-F(t)$ is the survival function associated with a CDF $F(t)$, and $f(t) = dF(t)/dt$ is the failure probability function (rate) at time t. The survival function indicates the amount of VMs that have survived at time t. The hazard rate can also be directly expressed in terms of the CDF as follows: $1 - F(t) = \exp\int_0^t -\lambda(x)\,dx$. The exponential distribution has a constant hazard rate λ. The Gompertz-Makeham distribution has an increasing failure rate to account for the increase in mortality, and its hazard rate is accordingly non-uniform and given by $\lambda(t) = \lambda + \alpha e^{\beta t}$.

Since we model multiple failure rates and deadline-driven preemptions, our hazard rate is expected to increase with time. Defining the survival function for our model: $S = 1 - \mathscr{F}$, and using Eq. 5 yields the hazard rate associated with our model:

$$\lambda = \frac{r_1 e^{-r_1 t} + r_2 e^{r_2(t-b)}}{1/A - 1 + e^{-r_1 t} - e^{r_2(t-b)}} \tag{6}$$

where we have introduced $r_1 = 1/\tau_1$, $r_2 = 1/\tau_2$ to denote the rates of preemptions associated with initial and final phases respectively.

Recall that the sharp increase in preemption rate only happens close to the deadline, which means that $b \lesssim L$. Thus, when $0 < t \ll b$, we get $\lambda(t) \approx r_1$, mimicking the hazard rate for the classic exponential distribution. As t approaches and exceeds b (i.e., $b \lesssim t < L$), the increase in the hazard rate due to the second failure process kicks in, accounting for the deadline-driven rise in preemptions. Note that our hazard rate satisfies $\lambda(t) \geq 0$ for $0 < t < L$.

3.3 Insights on the bathtub shape distribution

For constrained preemptions, one might expect to see uniformly distributed preemptions with a probability $1/L$ over $[0, L]$. However, as our empirical analysis shows, the preemption distribution is bathtub shaped. Interestingly, we can show using exact analytical arguments that non-uniform, bathtub distributions are in fact a *general* characteristic of systems with constrained preemptions, modulo some assumptions.

Lemma 1. *Consider N randomly distributed preemptions over an interval $[0, L]$. Assume that each preemption takes $w > 0$ time-units to perform, and preemptions cannot overlap, i.e, they occur in a mutually exclusive manner. Then, there exists $\epsilon > 0$ such that $P(L - \epsilon) > \frac{1}{L}$, where $P(t)$ is the probability of finding a preemption at time t.*

Proof. We first make some preliminary remarks and introduce concepts necessary to complete the proof.

Firstly, mutual exclusion of preemptions implies that there is a finite non-zero waiting time $w > 0$ between preemptions. For N preemptions to occur within L interval,

evidently, we must have $Nw < L$. Also, while $w > 0$, the time to perform the preemption is generally expected to be much smaller than the total time interval L. N preemptions occupy a "temporal volume" of Nw (volume here represents the one-dimensional volume). We assume that while a preemption may start at $t = 0$, the last preemption must finish by $t = L$. Thus, the amount of free or excluded "temporal volume" available within the constrained system is $L_e = L - w - (N-1)w = L - Nw$. The idea of excluded volume is central in physics and materials engineering where it underpins the origin of entropic or steric forces in material systems [39], [34].

Secondly, we note that the system of N preemptions within a constrained deadline of interval L maps *exactly* to a well known and analytically solvable system in classical statistical mechanics, the Tonks gas model [58], where one considers a system of N hard-spheres of diameter w to move along a line segment of length L. The structural quantities associated with this system including the probability of finding a sphere at position x within the interval L are computed by evaluating the partition function of the system, which essentially measures the number of valid system configurations [39]. Employing this mapping and the associated statistical mechanics tools, the original model of non-overlapping (interacting) preemptions can be mapped to a system of N overlapping (non-interacting) preemptions, each allowed to access an excluded volume of L_e, and the number of valid configurations is given by the partition function $Z_N = L_e^N$. For the case of N preemptions, we have $Z_N = (L - Nw)^N$.

We are interested in calculating the probability that a preemption starts at time $t = L-w$, i.e., $P(L-w)$. Given that the time to perform the preemption w is generally expected to be much smaller than the total time interval L, $P(L-w)$ is the probability of finding a preemption near the deadline. The assumption of mutually exclusive preemptions implies that no other preemption can be found for $t > L-w$, that is, $P(t > L-w) = 0$. Hence, the remaining $N-1$ preemptions must occur such that the last of those finish by $t = L-w$ (the preemption at time $L-w$ essentially sets an effective deadline for the other $N-1$ preemptions). The number of ways this can happen is given by the partition function $Z_{N-1} = L_e^{N-1} = (L-2w-(N-2)w)^{N-1} = (L-Nw)^{N-1}$, where $L_e = L-Nw$ is the corresponding excluded temporal volume accessible to each of the $N-1$ preemptions. It is interesting to note that this excluded volume is the same as that of the original N preemption system: this fortuitous result arises because the decrease in available volume to place the preemptions is commensurate with the need to place $N-1$ preemptions instead of N.

The probability $P(L-w)$ is obtained as the ratio of the valid configurations given by the two partition functions computed above. That is, $P(L-w) = Z_{N-1}/Z_N = \frac{1}{L-Nw} > \frac{1}{L}$, since $N \geq 1$ and $w > 0$. Choosing $\epsilon = w > 0$ completes the proof. □

By symmetry arguments, the above lemma is in fact valid for both the end points of the interval, i.e., $P(\epsilon) > \frac{1}{L}$. Thus, the probability of preemption is higher near the end points (deadline) than the average preemption probability of $1/L$, and we get a bathtub shaped distribution. Thus, the bathtub distribution can be considered to be a general artifact of constrained preemptions. Of course, the empirical preemption distribution is determined by the cloud platform's policies and supply and demand, and we elaborate more about the generality of our model and observation in Section 8.

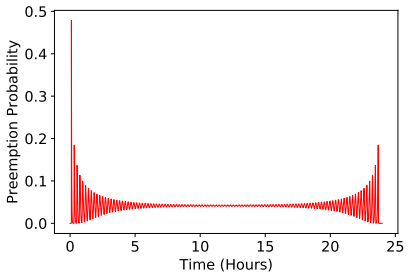

Fig. 2: Preemption probability with statistical mechanics partition function is also bathtub shaped.

For the above proof, we assumed that each preemption event occurs over a timespan of w, which is determined by the preemption warning that the cloud platform provides (which is 30 seconds for Google Preemptible VMs and 120 seconds for Amazon EC2 spot instances). Preempting a VM and reclaiming its resources involves manipulating the cluster-management state, and mutually exclusive preemptions may be convenient for cluster management, since serializing VM preemptions makes accounting and other cluster operations easier. From an application standpoint, non-overlapping preemptions are also beneficial, since handling multiple concurrent preemptions is significantly more challenging [48].

4 Application Policies For Constrained Preemptions

Having analyzed the statistical behavior of constrained preemptions and presented our probability model, we now examine how the bathtub shape of the failure rate impacts applications. Based on insights drawn from our statistical analysis and the model, we develop various policies for ameliorating the effects of preemptions. Prior work in transient computing has established the benefits of such policies for a broad range of applications. However, the constrained nature of preemptions introduces new challenges that do not arise in other transient computing environments such as Amazon EC2 spot instances, and thus new approaches are required. In this section, we first analyze the impact of constrained preemptions on job running time, and then develop new constrained-preemption aware policies for job scheduling and checkpointing. We will focus on long-running batch jobs that arise in many applications such as scientific computing. Extensions of our models and policies to distributed applications with different failure semantics is part of our future work.

4.1 Impact On Running Time

We now look at how temporally constrained preemptions impact the total expected running time of applications by using our failure probability model. When a preemption occurs during the job's execution, it results in wasted work, assuming there is no checkpointing. This increases the job's total expected running time, since it must restart after a

preemption. The expected wasted work depends on two factors:

1) The probability of the job being preempted during its execution.
2) *When* the preemption occurs during the execution.

We can analyze the wasted work due a preemption using the failure probability model. We first compute the expected amount of wasted work *assuming* the job faces a single preemption, which we denote by $E[W_1(T)]$, where T is the original job running time (without preemptions).

$$E[W_1(T)] = \int_0^T t\ P(t|t \leq T)\ dt, \tag{7}$$

where $P(t|t \leq T) = P(t)/P(t \leq T)$. Here, $P(t \leq T)$ is the probability that there is a preemption within time T and is given by $P(t \leq T) = F(T)$ where $F(T)$ is the CDF. $P(t)$ is the probability of a preemption at time t, and is given by $P(t) = f(t)$, where $f(t)$ is the probability distribution function given by Equation 2. We can therefore write the above equation as:

$$E[W_1(T)] = \int_0^T t\ P(t|t \leq T)\ dt = \frac{1}{F(T)} \int_0^T t\ f(t)\ dt. \tag{8}$$

We note that the integral is the same as the "expected lifetime", given by Equation 4. The above expression for the expected waste given a single preemption can be used by users and application frameworks to estimate the increase in running time due to preemptions. The total running time (also known as makespan) of a job *with* preemptions is given by:

$$E[T] = P(\text{no failure})\ T + P(1\text{ failure})\ \ (T + E[W_1(T)])\,, \tag{9}$$

where $P(\text{no failure}) = P(t > T) = 1 - F(T)$ and $P(1\text{ failure}) = P(t \leq T) = F(T)$. Expanding out these terms and using Equation 8, we get

$$E[T] = (1 - F\,(T))\,T + F(T)\,(T + E[W_1(T)]) = T + \int_0^T t\,f(t)\,dt. \tag{10}$$

This expression for the expected running time assumes that the job will be preempted at most once. An expression which considers the higher order terms and multiple job failures easily follows from the base case, but presents relatively low practical value.

Consequences for applications: Based on our analysis, both the increase in wasted time ($E[W_1(T)]/T$) and expected running time ($E[T]/T$) depend on the length of the job for non-memoryless constrained preemptions. For memoryless exponential distributions, the expected waste is simply $T/2$, but this assumption is not valid for constrained preemptions, and thus job lengths must be considered when evaluating the suitability of Preemptible VMs.

Users and transient computing systems can use the expected running time analysis for scheduling and monitoring purposes. Since the preemption characteristics are dependent on the type of the VM and temporal effects, this analysis also allows principled *selection* of VM types for jobs of a given length. For instance, VMs having a higher initial rate of preemptions are particularly detrimental for short jobs, because the jobs will see high rate of failure and are not long enough to run during the VM's stable period with low preemption rates. We evaluate the expected wasted time and running time for Google Preemptible VMs later in Section 6.

4.2 Job Scheduling and VM Reuse Policy

Our bathtub probability model also allows us to develop optimized job-scheduling policies for reducing job-failures. Many cloud-based applications and services are *long-running*, and typically run a continuous sequence of tasks and jobs on cloud VMs. In the case of deadline-constrained bathtub preemptions, applications face a choice: they can either run a new task on an already running VM, or relinquish the VM and run the task on a *new* VM. This choice is important in the case of non-uniform failure rates, since the job's failure probability depends on the "age" of the VM. Because of the bathtub failure distribution, VMs enjoy a long period of low failure rates during the middle of their total lifespan. Thus, it is beneficial to *reuse* VMs for multiple jobs, and relinquishing VMs after every job completion may not be an optimal choice.

However, jobs launched towards the end of VM life face a tradeoff. While they may start during periods of low failure rate, the 24 hour deadline-imposed sharp increase in preemptions poses a high risk of preemptions, especially for longer jobs. The alternative is to discard the VM and run the job on a new VM. However, since newly launched VMs also have high preemption rates (and thus high job failure probability), the choice of running the job on an existing VM vs. a new VM is not obvious.

Our job scheduling policy uses the preemption model to determine the preemption probability of jobs of a given length T. Assume that the running VM's age (time since launch) is s. The intuition is to reuse the VM only if the expected running time is lower, compared to running on a new VM. To compute the expected running time of a job of length T starting at vm-age s, we modify our earlier expression for running time (Equation 10) to:

$$E[T_s] = T + \int_s^{s+T} t\ f(t)\ dt \tag{11}$$

The alternative is to discard the VM and launch a new VM, in which case the expected running time is $E[T_0]$. Our job-scheduling policy is simple: When a job of running time T attempts to start on a VM of age s, if $E[T_s] \leq E[T_0]$, then we run the job on the existing VM. Otherwise, a new VM is launched.

This technique can also be used to find the job length T^*, when the transition occurs between re-using and launching a new VM. Thus, accurate job lengths are not necessary, and we only need to know whether $T < T^*$, and only a rough estimate of the job lengths is required. We assume that because most scientific computing workloads involve exploration of some parameter space, they often have homogenenous running times.

Simple Job Scheduling Policy.

4.3 VM Selection

VM-selection is an important optimization in cloud environments, because VMs have different tradeoffs of cost, performance, and preemption characteristics. Application

performance is affected by the size of the VM (due to network communication and parallel scaling overheads), and the preemption rates. We propose to develop cost models for selecting the "right" type of VMs that minimizes the expected job failure probability and cost by using the analytical preemption models. Initial analysis indicates that careful VM selection can reduce costs by up to 30%.

We note that this search is different from conventional speedup plots in which the objective is to determine how well an application scales with increasing amount of resources and parallelism. In contrast, we *fix* the total amount of resources allocated to the application's job ($= \mathcal{R}$), and only vary *how* these resources are distributed, which affects communication overhead and hence the performance. We assume that the total resource requirement for a job, $\mathcal{R}$, is provided by the user based on prior speedup data, the user's cloud budget, and the deadline for job completion.

Since server selection involves a tradeoff between cost, performance, and preemptions, we develop a model that allows us to optimize the resource allocation and pick the best VM type that minimizes the expected cost of running an application on Our service.

(Prateek: Highlight tradeoffs here. Larger servers better but more preemption.)

Let us assume that the cloud provider offers N server types, with the price (per unit time) of a server type equal to c_i. The overall expected cost of running a job can then be expressed as follows:

$$E[C_{(i,n_i)}] = n_i \times c_i \times E[\mathcal{T}_{(i,n_i)}]. \tag{12}$$

Here, $E[\mathcal{T}_{(i,n_i)}]$ denotes the expected turnaround time of the job (accounting for preemptions) on n_i servers of type i. This turnaround time depends on whether the job needs to be recomputed because of preemptions, and is expressed as:

$$E[\mathcal{T}_{(i,n_i)}] = T_{(i,n_i)} + E[\text{Recomputation Time}]. \tag{13}$$

Here, $T_{(i,n_i)}$ is the base running time of a job without preemptions, which we obtain empirically as explained in the previous subsection. Since jobs have to be rerun when they fail due to preemptions, the recomputation time is:

$$E[\text{Recomputation Time}] = \frac{T_{(i,n_i)}}{2} \times P(\text{at least one preemption}) \tag{14}$$

Our expression of the recomputation time is based on the common assumption that jobs will fail at the half-way mark on average [22], [17]. The probability that at least one VM out of n_i will be preempted during the job execution is:

$$P(\text{at least one preemption}) = 1 - P(\text{no preemptions}) \tag{15}$$

$$= 1 - \left(1 - P\left(i, T_{(i,n_i)}\right)\right)^{n_i}. \tag{16}$$

Here, $P(i, T_{(i,n_i)})$ denotes the probability of a preemption of a VM of type i when a job of duration $T_{(i,n_i)}$ runs on it. It depends on the type of server, and its associated expected lifetime, and is defined as:

$$P\left(i, T_{(i,n_i)}\right) = \min\left(\frac{T_{(i,n_i)}}{E[L_i]}, 1\right), \tag{17}$$

(Prateek: Can replace by CDF based failure prob)

where $E[L_i]$ is the expected lifetime of the VM of type i extracted using the preemption model (Equation 4). We also assume that the running time of *individual* jobs in a bag (T), will be smaller than the expected lifetime of the VMs, otherwise we will see no forward progress since the jobs will always be preempted before completion. This is a safe assumption, since more than 90% HPC jobs are less than 2 hours long (Figure 9 inset), and the expected lifetime of transient VMs is more than 10 hours. This restriction only applies to individual jobs—Our service can run large bags of jobs even if their total running time exceeds the VM lifetime by replenishing preempted VMs.

(Prateek: Policy is to use older more stable VMs where possible)

5 Implementing a Batch Computing Service For Preemptible VMs

We have implemented a prototype batch computing service that implements various policies for constrained preemptions. We use this service to examine the effectiveness and practicality of our model and policies in real-world settings. Our service is implemented as a light-weight, extensible framework that makes it convenient and cheap to run batch jobs in the cloud. We have implemented our prototype in Python in about 2,000 lines of code, and currently support running VMs on the Google Cloud Platform [4].

We use a centralized controller (Figure 3), which implements the VM selection and job scheduling policies described in Section 4. The controller can run on any machine (including the user's local machine, or inside a cloud VM), and exposes an HTTP API to end-users. Users submit jobs to the controller via the HTTP API, which then launches and maintains a cluster of cloud VMs, and maintains the job queue and metadata in a local database.

Our service integrates, and interfaces with two primary services. First, it uses the Google cloud API [3] for launching, terminating, and monitoring VMs. Once a cluster is launched, it then configures a cluster manager such as Slurm [8] or Torque [10], to which it submits jobs. Our service uses the Slurm cluster manager, with each VM acting as a Slurm "cloud" node, which allows Slurm to gracefully handle VM preemptions. The Slurm master node runs on a small, 2 CPU non-preemptible VM, which is shared by all applications and users. We monitor job completions and failures (due to VM preemptions) through the use of Slurm call-backs, which issue HTTP requests back to the central service controller.

Policy Implementation: Our service creates and manages clusters of transient cloud servers, manages all aspects of the VM lifecycle and costs, and implements the model-based policies. It manages a cluster of VMs, and parametrizes the bathtub model based on the VM type, region, time-of-day, and day-of-week. When a new batch job is to be launched, we find a "free" VM in the cluster that is idle, and uses the job scheduling policy to determine if the VM is suitable or a new VM must be launched. Due to the bathtub nature of the failure rate, VMs that have survived the initial failures are "stable" and have a very low rate of failure, and thus are "valuable". We keep these stable VMs as "hot spares" instead of terminating them, for a period of one hour. For the checkpointing policy, our dynamic programming algorithm

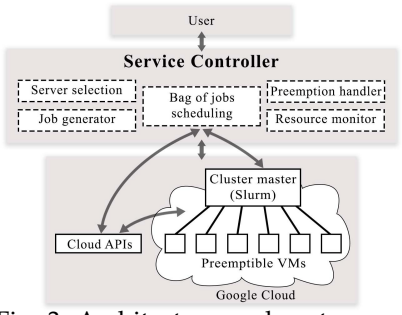

Fig. 3: Architecture and system components of our batch computing service.

has a time complexity of $O(T^3)$, for a job of length T. To minimize this overhead, we precompute the checkpointing schedule of jobs of different lengths, and don't need to compute the checkpoint schedule for every new job.

Bag of Jobs Abstraction For Scientific Simulations: While our service is intended for general batch jobs, we incorporate a special optimization for scientific simulation workloads that improves the ease-of-use of our service, and also helps in our policy implementation. Our insight is that most scientific simulations involve launching a series of jobs that explore a large parameter space that results from different combinations of physical and computational parameters. These workloads can be abstracted as a "bag of jobs", with each job running the same application with different parameters. A bag of jobs is characterized by the job and all the different parameters with which it must be executed. Within a bag, jobs show little variation in their running time and execution characteristics.

We allow users to submit entire bags of jobs, which permits us to determine the running time of jobs based on previous jobs in the bag. For constrained preemptions, the running time and checkpointing are determined by job lengths, and the job run time estimates are extremely useful. Having a large sequence of jobs is also particularly useful with bathtub preemptions, since we can re-use "stable" VMs with low preemption probability for running new jobs from a bag. If jobs were submitted one at a time, a batch computing service may have to terminate the VM after job completion, which would increase the job failure probability resulting from running on new VMs that have a high initial failure rate.

6 Model and Policy Evaluation

In this section, we present analytical and empirical evaluation of constrained preemptions. We have already presented the statistical analysis of our model in Section 3, and we now focus on answering the following questions:

1) How do constrained preemptions impact the total running time of applications?
2) What is the effect of our model-based policies when compared to existing transient computing approaches?
3) What is the cost and performance of our batch computing service for real-world workloads?

Environment and Workloads: All our empirical evaluation is conducted on the Google Cloud Platform using our batch computing service described in Section 5. All the experiments are conducted in the same time period, and have the same preemption characteristics, as described in our data collection methodology in Section 3. We use three scientific computing workloads that are representative of applications in the broad domains of physics and material sciences:

Nanoconfinement. The nanoconfinement application launches molecular dynamics (MD) simulations of ions in nanoscale confinement created by material surfaces [34], [36]. The running time is 14 minutes on a 64 CPU core cluster (4 n1-highcpu-16 VMs).

Shapes. The Shapes application runs an MD-based optimization dynamics to predict the optimal shape of deformable, charged nanoparticles [31], [18]. The running time is 9 minutes on a 64 CPU core cluster (4 n1-highcpu-16 VMs).

LULESH. Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics is a popular benchmark for hydrodynamics simulations of continuum material models [37], [38]. The running time is 12.5 minutes on 8 n1-highcpu-8 VMs.

6.1 Model-based Policies

We now evaluate the effectiveness of model-driven policies that we proposed earlier in Section 4. Wherever applicable, we compare against policies designed for EC2 spot instances [28], [55] that have memoryless preemptions. However we also note that certain resource management challenges such as the preemption-rate aware job scheduling are *inherent* to constrained preemptions, and no existing equivalent policies can be found for memoryless techniques.

6.1.1 Job Scheduling

Previously, we have quantified the increase in running time due to preemptions, but we had assumed that jobs start on a newly launched server. In many scenarios however, a server may be used for running a long-running sequence of jobs, such as in a batch-computing service. Our job scheduling policy is model-driven and decides whether to request a new VM for a job or run it on an existing VM. A new VM may be preferable if the job starts running near the VM's 24 hour preemption deadline.

Figure 4 shows the effect of our job scheduling policy for a six hour job, for different job starting times (relative to the VM's starting time). We compare against a baseline of memoryless job scheduling that is not informed by constrained preemption dynamics. Such memoryless policies are the default in existing transient computing systems such as SpotOn [55]. In the absence of insights about bathtub preemptions, the memoryless policy continues to run jobs on the existing VM. As the figure shows, the empirical job failure probability is bathtub shaped. However since the job is 6 hours long, with the memoryless policy, it will always fail when launched after $24 - 6 = 18$ hours. In contrast, our model-based policy determines that after 18 hours, we will be better off running the job on a newer VM, and results in a constant lower job failure probability (=0.4). The failure probability is constant because the jobs will always be launched on a new VM after 18 hours, resulting in a failure probability at time=0. Thus, our model-based job scheduling policy can reduce job failure probability by taking into account the time-varying failure rates of VMs, which is not considered by existing systems that use memoryless scheduling policies.

Tipping points:

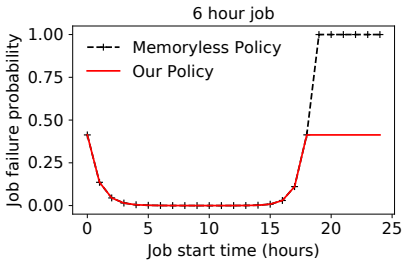

Fig. 4: Effect of job start time on the failure probability.

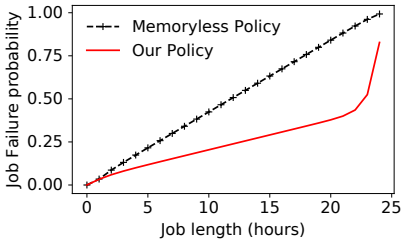

Fig. 5: Job failure probability for jobs of different lengths.

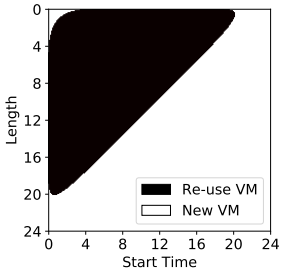

Fig. 6: Job lengths and starting points where VM should be reused (black), and a new VM should be launched (white)

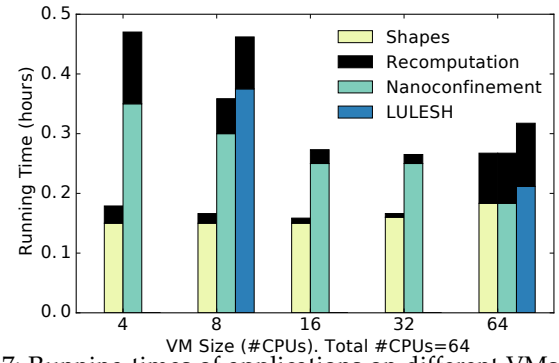

Fig. 7: Running times of applications on different VMs. Total number of CPUs is 64, yielding different number of VMs in each case. We see different tradeoffs in the base running times and recomputation times.

The job failure probability is determined by the job length and the job starting time. We examine the failure probability for jobs of different lengths (uniformly distributed) in Figure 5, in which we average the failure probability across different start times. We again see that our policy results in significantly lower failure probability compared to memoryless scheduling. For all but the shortest and longest jobs, the failure probability with our policy is *half* of that of existing memoryless policies. This reduction is primarily due to how the two policies perform for jobs launched near the end of the VM preemption deadline, which we examined previously in Figure 4.

6.1.2 Impact of server exploration

When an application (i.e., bag of jobs) requests a total number of CPUs to run each of its jobs, Our service first runs its exploration phase to find the "right" VM for the application. Our service searches for the VM that minimizes the total expected cost $E[C_{(i,n_i)}]$ of running the application. Thus, even if the *total* amount of resources (i.e., number of CPUs) per job is held constant, the total running time (i.e., turnaround time) of an application depends on the choice of the VM type (i), and the associated number of VMs (n_i) required to meet the allocation constraint (Section **??**). With preemptible instances, the total running time of a job is composed of two factors: the "base" running time of the job without any preemptions ($T_{(i,n_i)}$), and the expected recomputation time which depends on the probability of job failure (Equation 14).

Figure 7 shows the running times of the Nanoconfinement, Shapes, and LULESH applications, when they are deployed on different VM sizes. In all cases, the total number of CPUs per job is set to 64, and thus the different VM sizes yield different cluster sizes (e.g., 16 VMs with 4 CPUs or 32 VMs with 2 CPUs). LULESH requires CPUs to be cube of an integer, which limits the valid cluster configurations.

For Nanoconfinement and LULESH, we observe that the base running times (without preemptions) reduce when moving to larger VMs, because this entails lower communication costs. For Nanoconfinement, the running time on the "best" VM (i.e., with 32 CPUs) is nearly 40% lower as compared to the worst case. On the other hand, the Shapes application can scale to a larger number of VMs without any significant communication overheads, and does not see any significant change in its running time.

Figure 7 also shows the expected turnaround time $E[\mathcal{T}_{(i,n_i)}]$, that is obtained by adding the the expected recomputation time, which depends on the expected lifetimes of the VM and the number of VMs, and is computed using the cost model introduced in Section **??**. While selecting larger VMs may reduce communication overheads and thus improve performance, it is not an adequate policy in the case of preemptible VMs, since the preemptions can significantly increase the turnaround time. Therefore, even though the base running time of Nanoconfinement is lower on a 64 CPU VM, the recomputation time on the 64 CPU VM is almost $4\times$ higher compared to a 2x32-CPU cluster, due to the much lower expected lifetime of the larger VMs. Thus, on preemptible servers, there is a tradeoff between the base running time which only considers parallelization overheads, and the recomputation time. By considering *both* these factors, Our service's server selection policy can select the best VM for an application.

Result: *SciSpot's server selection, by considering both the base running time and recomputation time, can improve performance by up to 40% , and can keep the increase in running time due to recomputation to less than 5%.*

6.1.3 Cost

The primary motivation for using preemptible VMs is their significantly lower cost compared to conventional "on-demand" cloud VMs that are non-preemptible. Figure 8 compares the cost of running different applications with different cloud VM deployments. Our service, which uses both cost-minimizing server selection, and preemptible VMs, results

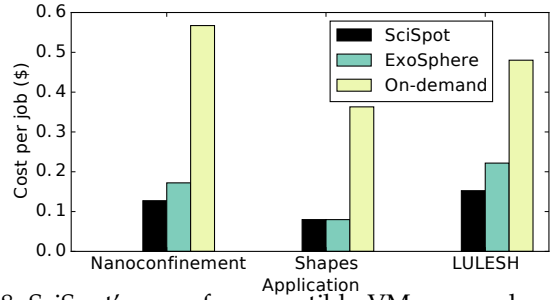

Fig. 8: SciSpot's use of preemptible VMs can reduce costs by up to $5\times$ compared to conventional cloud deployments, and 20% compared to the state of the art EC2 spot instance selection (ExoSphere [48]).

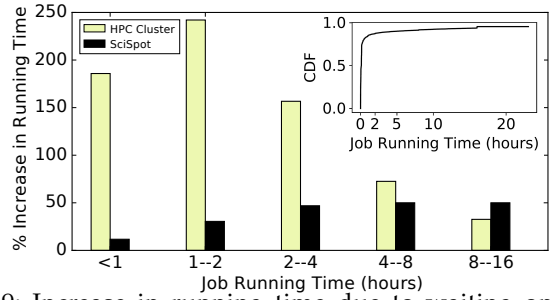

Fig. 9: Increase in running time due to waiting on HPC clusters is significantly higher than the recomputation time for Our service, except for very long and rare jobs (see inset).

in significantly lower costs across the board, even when accounting for preemptions and recomputations. We also compare against ExoSphere [48], a state of the art system for transient server selection. ExoSphere implements a portfolio-theory approach using EC2 spot prices to balance average cost saving and risk of revocations using diversification and selecting VMs with low price correlation. However, this approach is ineffective for the flat prices of Google Preemptible VMs. Unlike Our service, ExoSphere does *not* consider application performance when selecting servers, and thus is unable to select the best server for parallel applications. Since the Google `highcpu` VMs have the same price per CPU, ExoSphere picks an arbitrary "median" VM to break ties, which may not necessarily yield the lowest running times. This results in 20% cost increase over Our service.

Result: *SciSpot reduces computing costs by up to $5\times$ compared to conventional on-demand cloud deployments.*

6.1.4 Comparison with HPC Overhead

Scientific computing applications are typically run on large-scale HPC clusters, where different performance and cost dynamics apply. While there are hardware differences between cloud VMs and HPC clusters that can contribute to performance differences, we are interested in the performance "overheads". In the case of Our service, the job failures and recomputations increase the job turnaround time, and are thus the main source of overhead.

On HPC clusters, jobs enjoy significantly lower recomputation probability, since the hardware on these clusters has MTTFs in the range of years to centuries [23]. However, we emphasize that there exist *other* sources of performance overheads in HPC clusters. In particular, since HPC clusters have high resource utilization, they also have significant *waiting* times. On the other hand, cloud resource utilization is low [60] and there is usually no need to wait for resources, which is why transient servers exist in the first place.

Thus, we compare the performance overhead due to preemptions for Our service, and job waiting times in conventional HPC deployments. To obtain the job waiting times in HPC clusters, we use the LANL Mustang traces published as part of the Atlas trace repository [12]. We analyze the waiting time of over two million jobs submitted over a 5 year period, and compute the increase in running time of the job due to the job waiting or queuing time.

Figure 9 compares the overhead (as percentage increase in running time) of Our service and HPC clusters for jobs of different lengths. We see that the average performance overhead due to waiting can be significant in the case of HPC clusters, and the job submission latency and queuing time dominate for smaller jobs, increasing their total turnaround time by more $2.5\times$. This waiting is amortized in the case of longer running jobs, and the overhead for longer jobs is around 30%.

On the other hand, Our service's performance overhead is significantly smaller for jobs of up to 8 hours in length. For longer jobs, the limited lifetime of Google Preemptible VMs (24 hours) begins to significantly increase the preemption probability and expected recomputation time. We emphasize that these are *individual* job lengths, and not the running time of entire bag of jobs. We note that these large single jobs are rare, accounting for less than 5% of all HPC jobs (see inset in Figure 9). For smaller jobs (within a much larger bag), both the preemption probability and recomputation overhead is much smaller.

Result: *Our service's overhead of recomputation due to preemptions is small, and is up to $10\times$ lower compared to the overhead of waiting in conventional HPC clusters.*

7 Related Work

Transient Cloud Computing. The significantly lower cost of spot instances makes them attractive for running preemption and delay tolerant batch jobs [55], [32], [67], [62], [40], [19], [24], [59], [28]. The challenges posed by Amazon EC2 spot instances, the first transient cloud servers, have received significant attention from both academia and industry [9]. The distinguishing characteristic of EC2 spot instances is their dynamic auction-based pricing, and choosing the "right" bid price to minimize cost and performance degradation is the focus of much of the past work on transient computing [33], [42], [57], [61], [66], [70], [68], [71], [54], [64], [27]. However, it remains to be seen how Amazon's recent change [11], [30], [14], [45] in the preemption model of spot instances affects prior work. Non-price based transient availability models, such as temporally constrained preemptions, have received scant attention due to the difficulty in obtaining empirical preemption data—which we hope our dataset remedies.

Preemption Mitigation. Effective use of transient servers usually entails the use of fault-tolerance techniques such as checkpointing [46], migration [49], and replication [55]. In the context of HPC workloads, [41], [26], [56] develop checkpointing and bidding strategies for MPI applications running on EC2 spot instances. However, periodic checkpointing [23], [17] is not optimal in our case because preemptions are not memoryless.

Preemption Modeling. Conventionally, exponential distribution have been used to model preemptions, even for EC2 spot instances [71], [46], [47]. Our preemption model provides a novel characterization of bathtub shaped failure rates not captured even by Weibull distributions, and is distinct from prior efforts [43], [21]. Recent work [52] has also found evidence of the bath-tub failure distribution for Google Preemptible GPU VMs, and confirms our observations.

8 Discussion and Future Directions

Constrained preemptions are a relatively unexplored phenomenon and challenging to model. Our model and the associated data expand transient cloud computing to beyond EC2-spot. However, many questions and avenues of future investigation remain open:

What if preemption characteristics change? Our model allows detecting policy and phase changes by comparing observed data with model-predictions and detect change-points, and a long-running cloud service can continuously update the model based on recent preemption behavior. However, changes are rare: Google's preemption policy has not changed since its inception in 2015. Regardless, VMs with constrained preemptions are an interesting *new* type of transient resource, and our analysis, observations, and policies should continue to be relevant. We have also shown that our policies are not particularly sensitive to the model parameters, and even using a "wrong" or outdated model can provide significant benefits compared to existing memoryless models. Our modeling approach works across a wide range of instance types and is able to model CDFs of instances with both very high and very low failure rates, and thus is general. Moreover, because bathtub preemptions are good for the applications, they will continue to remain a good choice for constrained preemptions making our approach generalizable to other system environments beyond the Google cloud computing systems. Finally, the principle adopted to break down the problem into the superposition of processes characterized by different failure rates can also be considered as a general framework to understand and guide policies for mitigating preemption-induced effects in other cloud environments.

***Phase-wise* model.** Our statistical analysis indicates that the preemption rates have three distinct phases. The analytical model derived in this work is continuously differentiable and allows capturing the three phases reasonably well. It may be possible to use a "phase-wise" model such as a piece-wise continuously differentiable model, where the three phases are modeled either as segmented linear regions (found using segmented linear regression), or an initial exponential phase and two linear phases. Such a piece-wise model could capture the phase transitions with even more accuracy. Our analytical model informed by empirical data and based on well-defined assumptions can guide the development of such simpler heuristics and modeling approaches, as well as the interpretation of their results. The analytical model also provides a measure to distill the contributions of effects ignored in its derivation (e.g., only 2 failure rates) in changing the VM expected lifetimes and checkpointing policies. At the same time, it also provides a principled approach to extend the model to more complex cases, e.g., systems characterized by processes with more than 2 failure rates.

Acknowledgments. We wish to thank all the anonymous reviewers and our shepherd Ali R. Butt, for their insightful comments and feedback. This research was supported by Google cloud credits for research. V.J. was partially supported by NSF through Award DMR-1753182.

References

[1] Alibaba Cloud Preemptible Instances. https://www.alibabacloud.com/help/doc-detail/52088.htm .

[2] Amazon EC2 Spot Instances, howpublished=https://aws.amazon.com/ec2/spot/ ,.

[3] Google Cloud API Documentation. https://cloud.google.com/apis/docs/overview .

[4] Google Cloud Platform. https://cloud.google.com/ .

[5] Google Cloud Preemptible VM Instances Documentation, howpublished=https://cloud.google.com/compute/docs/instances/preemptible,.

[6] Packet Spot Market. https://support.packet.com/kb/articles/spot-market .

[7] Scipy curve fit documentation. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve$_fit$.

[8] Slurm Workload Manager. https://slurm.schedmd.com/documentation.html .

[9] Spotinst. https://spotinst.com/.

[10] Torque Resource Manager. http://www.adaptivecomputing.com/products/torqu

[11] New Amazon EC2 Spot pricing model. https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/ , March 2018.

[12] Amvrosiadis, G., Park, J. W., Ganger, G. R., Gibson, G. A., Baseman, E., and DeBardeleben, N. On the diversity of cluster workloads and its impact on research results. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)* (2018), pp. 533–546.

[13] Azure Low-priority Batch VMs. https://docs.microsoft.com/en-us/azure/batch/batch-low-pri-vms .

[14] Baughman, M., Caton, S., Haas, C., Chard, R., Wolski, R., Foster, I., and Chard, K. Deconstructing the 2017 changes to aws spot market pricing. In *Proceedings of the 10th Workshop on Scientific Cloud Computing* (2019), ACM, pp. 19–26.

[15] Baughman, M., Haas, C., Wolski, R., Foster, I., and Chard, K. Predicting amazon spot prices with lstm networks. In *Proceedings of the 9th Workshop on Scientific Cloud Computing* (2018), ACM, p. 1.

[16] Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. Deconstructing Amazon EC2 Spot Instance Pricing. *ACM TEC 1*, 3 (September 2013).

[17] Bougeret, M., Casanova, H., Rabie, M., Robert, Y., and Vivien, F. Checkpointing strategies for parallel jobs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11* (Seattle, Washington, 2011), ACM Press, p. 1.

[18] Brunk, N. E., and Jadhao, V. Computational studies of shape control of charged deformable nanocontainers. *Journal of Materials Chemistry B* (2019).

[19] Chohan, N., Castillo, C., Spreitzer, M., Steinder, M., Tantawi, A., and Krintz, C. See Spot Run: Using Spot Instances for MapReduce Workflows. In *HotCloud* (June 2010).

[20] Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, ACM, pp. 153–167.

[21] Crevecoeur, G. A model for the integrity assessment of ageing repairable systems. *IEEE Transactions on reliability 42*, 1 (1993), 148–155.

[22] Daly, J. T. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. *Future Generation Computer Systems 22*, 3 (2006).

[23] Dongarra, J., Herault, T., and Robert, Y. Fault tolerance techniques for high-performance computing. 66.

[24] Dubois, D. J., and Casale, G. Optispot: minimizing application deployment cost using spot cloud resources. *Cluster Computing* (2016), 1–17.

[25] GHIT, B., AND EPEMA, D. Better safe than sorry: Grappling with failures of in-memory data analytics frameworks. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2017), HPDC '17, ACM, pp. 105–116.

[26] GONG, Y., HE, B., AND ZHOU, A. C. Monetary cost optimizations for MPI-based HPC applications on Amazon clouds: checkpoints and replicated execution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15* (Austin, Texas, 2015), ACM Press, pp. 1–12.

[27] GUO, W., CHEN, K., WU, Y., AND ZHENG, W. Bidding for Highly Available Services with Low Price in Spot Instance Market. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '15* (Portland, Oregon, USA, 2015), ACM Press, pp. 191–202.

[28] HARLAP, A., CHUNG, A., TUMANOV, A., GANGER, G. R., AND GIBBONS, P. B. Tributary: spot-dancing for elastic services with latency slos. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)* (2018), pp. 1–14.

[29] HARLAP, A., TUMANOV, A., CHUNG, A., GANGER, G. R., AND GIBBONS, P. B. Proteus: Agile ml elasticity through tiered reliability in dynamic resource markets. In *Proceedings of the Twelfth European Conference on Computer Systems* (New York, NY, USA, 2017), EuroSys '17, ACM, pp. 589–604.

[30] IRWIN, D., SHENOY, P., AMBATI, P., SHARMA, P., SHASTRI, S., AND ALI-ELDIN, A. The price is (not) right: Reflections on pricing for transient cloud servers. *2019 28th International Conference on Computer Communication and Networks (ICCCN)* (2019), 1–9.

[31] JADHAO, V., THOMAS, C. K., AND OLVERA DE LA CRUZ, M. Electrostatics-driven shape transitions in soft shells. *Proceedings of the National Academy of Sciences 111*, 35 (2014), 12673–12678.

[32] JAIN, N., MENACHE, I., AND SHAMIR, O. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In *11th International Conference on Autonomic Computing (ICAC 14)*, USENIX Association.

[33] JAVADI, B., THULASIRAM, R., AND BUYYA, R. Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In *UCC* (December 2011).

[34] JING, Y., JADHAO, V., ZWANIKKEN, J. W., AND OLVERA DE LA CRUZ, M. Ionic structure in liquids confined by dielectric interfaces. *The Journal of chemical physics 143*, 19 (2015), 194508.

[35] JOAQUIM, P., BRAVO, M., RODRIGUES, L., AND MATOS, M. Hourglass: Leveraging transient resources for time-constrained graph processing in the cloud. In *Proceedings of the Fourteenth EuroSys Conference 2019* (New York, NY, USA, 2019), EuroSys '19, ACM, pp. 35:1–35:16.

[36] KADUPITIYA, J., MARRU, S., FOX, G. C., AND JADHAO, V. Ions in nanoconfinement, Dec 2017. Online on nanoHUB; source code on GitHub at github.com/softmaterialslab/nanoconfinement-md.

[37] KARLIN, I., BHATELE, A., KEASLER, J., CHAMBERLAIN, B. L., COHEN, J., DEVITO, Z., HAQUE, R., LANEY, D., LUKE, E., WANG, F., RICHARDS, D., SCHULZ, M., AND STILL, C. Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)* (Boston, USA, May 2013).

[38] KARLIN, I., KEASLER, J., AND NEELY, R. Lulesh 2.0 updates and changes. Tech. Rep. LLNL-TR-641973, August 2013.

[39] KRAUTH, W. *Statistical mechanics: algorithms and computations*, vol. 13. OUP Oxford, 2006.

[40] LIU, H. Cutting MapReduce Cost with Spot Market. In *HotCloud* (June 2011).

[41] MARATHE, A., HARRIS, R., LOWENTHAL, D., DE SUPINSKI, B. R., ROUNTREE, B., AND SCHULZ, M. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *HPDC* (2014), ACM.

[42] MIHAILESCU, M., AND TEO, Y. M. The Impact of User Rationality in Federated Clouds. In *CCGrid* (2012).

[43] MUDHOLKAR, G. S., AND SRIVASTAVA, D. K. Exponentiated weibull family for analyzing bathtub failure-rate data. *IEEE transactions on reliability 42*, 2 (1993), 299–302.

[44] OUYANG, X., IRWIN, D., AND SHENOY, P. Spotlight: An information service for the cloud. In *IEEE International Conference on Distributed Computing Systems (ICDCS)* (2016).

[45] PHAM, T.-P., RISTOV, S., AND FAHRINGER, T. Performance and behavior characterization of amazon ec2 spot instances. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (2018), IEEE, pp. 73–81.

[46] SHARMA, P., GUO, T., HE, X., IRWIN, D., AND SHENOY, P. Flint: Batch-Interactive Data-Intensive Processing on Transient Servers. In *EuroSys* (April 2016).

[47] SHARMA, P., IRWIN, D., AND SHENOY, P. How Not to Bid the Cloud. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)* (June 2016), USENIX.

[48] SHARMA, P., IRWIN, D., AND SHENOY, P. Portfolio-driven resource management for transient cloud servers. In *Proceedings of ACM Measurement and Analysis of Computer Systems* (June 2017), vol. 1, p. 23.

[49] SHARMA, P., LEE, S., GUO, T., IRWIN, D., AND SHENOY, P. SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market. In *EuroSys* (April 2015).

[50] SHASTRI, S., AND IRWIN, D. Hotspot: automated server hopping in cloud spot markets. In *Proceedings of the 2017 Symposium on Cloud Computing* (2017), ACM, pp. 493–505.

[51] SHASTRI, S., RIZK, A., AND IRWIN, D. Transient guarantees: Maximizing the value of idle cloud capacity. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Piscataway, NJ, USA, 2016), SC '16, IEEE Press, pp. 85:1–85:11.

[52] SHIJIAN LI, R. W., AND GUO, T. Characterizing and modeling distributed training with transient cloud gpu servers. *40th IEEE International Conference on Distributed Computing Systems (ICDCS'20)*.

[53] SINGH, R., SHARMA, P., IRWIN, D., SHENOY, P., AND RAMAKRISHNAN, K. Here Today, Gone Tomorrow: Exploiting Transient Servers in Data Centers. *IEEE Internet Computing 18*, 4 (July/August 2014).

[54] SONG, Y., ZAFER, M., AND LEE, K. Optimal Bidding in Spot Instance Market. In *Infocom* (March 2012).

[55] SUBRAMANYA, S., GUO, T., SHARMA, P., IRWIN, D., AND SHENOY, P. SpotOn: A Batch Computing Service for the Spot Market. In *SOCC* (August 2015).

[56] TAIFI, M., SHI, J. Y., AND KHREISHAH, A. SpotMPI: A Framework for Auction-Based HPC Computing Using Amazon Spot Instances. In *Algorithms and Architectures for Parallel Processing*, Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7017. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 109–120.

[57] TANG, S., YUAN, J., AND LI, X. Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance. In *CLOUD* (June 2012).

[58] TONKS, L. The complete equation of state of one, two and three-dimensional gases of hard elastic spheres. *Phys. Rev. 50* (Nov 1936), 955–963.

[59] VARSHNEY, P., AND SIMMHAN, Y. AutoBoT : Resilient and Cost-effective Scheduling of a Bag of Tasks on Spot VMs. *IEEE Transactions on Parallel and Distributed Systems* (2019), 1–1.

[60] VERMA, A., PEDROSA, L., KORUPOLU, M., OPPENHEIMER, D., TUNE, E., AND WILKES, J. Large-scale cluster management at google with borg. In *EuroSys* (2015), ACM.

[61] WEE, S. Debunking Real-Time Pricing in Cloud Computing. In *CCGrid* (May 2011).

[62] WIEDER, A., BHATOTIA, P., POST, A., AND RODRIGUES, R. Orchestrating the deployment of computations in the cloud with conductor. In *NSDI 12* (2012).

[63] WOLSKI, R., AND BREVIK, J. Providing statistical reliability guarantees in the aws spot tier. In *Proceedings of the 24th High Performance Computing Symposium* (2016), Society for Computer Simulation International, p. 13.

[64] WOLSKI, R., BREVIK, J., CHARD, R., AND CHARD, K. Probabilistic guarantees of execution duration for Amazon spot instances. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17* (Denver, Colorado, 2017), ACM Press, pp. 1–11.

[65] WOLSKI, R., BREVIK, J., CHARD, R., AND CHARD, K. Probabilistic guarantees of execution duration for amazon spot instances. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017), ACM, p. 18.

[66] XU, H., AND LI, B. A Study of Pricing for Cloud Resources. *Performance Evaluation Review 40*, 4 (March 2013).

[67] YI, S., KONDO, D., AND ANDRZEJAK, A. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* (2010), IEEE, pp. 236–243.

[68] ZAFER, M., SONG, Y., AND LEE, K. Optimal Bids for Spot VMs in a Cloud for Deadline Constrained Jobs. In *CLOUD* (2012).

[69] ZHANG, C., GUPTA, V., AND CHIEN, A. A. Information models: Creating and preserving value in volatile cloud resources. In *2019*

IEEE International Conference on Cloud Engineering (IC2E) (June 2019), pp. 45–55.

[70] ZHANG, Q., GÜRSES, E., BOUTABA, R., AND XIAO, J. Dynamic Resource Allocation for Spot Markets in Clouds. In *Hot-ICE* (March 2011).

[71] ZHENG, L., JOE-WONG, C., TAN, C. W., CHIANG, M., AND WANG, X. How to Bid the Cloud. In *SIGCOMM* (August 2015).